CLAIMS

1. A method for updating a filter engine opcode tree, comprising the following steps:

(a) compiling a new query to derive a series of opcode objects;

(b) traversing the opcode tree according to the series of opcode objects until an opcode object is encountered that is not included in the opcode tree, opcode objects being represented in the opcode tree as opcode nodes; and

(c) adding new opcode tree opcode nodes to correspond to the encountered opcode object and subsequent opcode objects in the series of opcode objects.

2. The method as recited in claim 1, wherein one or more of the steps are performed dynamically at runtime.

3. The method as recited in claim 1, further comprising performing steps (b) and (c) in a component of the filter engine.

4. The method as recited in claim 1, further comprising executing the opcode tree against an input to evaluate the new query and one or more other queries against the input.

5. The method as recited in claim 1, further comprising:

receiving a request to remove a first query from the opcode tree;

identifying one or more opcode nodes in the opcode tree that correspond to the first query;

removing any identified opcode node that does not correspond to a second query.

6. The method as recited in claim 1, further comprising updating a branch node in the opcode tree to add a reference to the new opcode nodes, the branch node being referenced from a parent opcode node that corresponds to a last opcode object from the series of opcode objects that was found in the traversal of the opcode tree.

7. The method as recited in claim 6, the branch node further comprising updating the branch node to include an indexed lookup routine that references several dependent opcode nodes that perform a similar function.

8. The method as recited in claim 7, further comprising analyzing opcode nodes that depend from the branch node and including the indexed lookup routine only if including the indexed lookup routine provides more efficient processing of the dependent nodes that a generic branch node processing routine.

9. The method as recited in claim 7, the indexed lookup routine further comprising one of the following routines: a hash routine; a routine that uses tries; an interval tree routine.

10. A filter engine stored on one or more computer-readable media, comprising:

a filter table that includes a plurality of queries, at least two of the queries including a common prefix;

a compiler configured to compile each query into a series of opcode blocks;

an opcode tree stored in memory and including opcode nodes that each correspond to an opcode block such that executing the opcode nodes evaluates the plurality of queries, at least one opcode node corresponding to an opcode block included in the common prefix; and

an opcode merger configured to merge a new query to the opcode tree by adding at least one opcode node that corresponds to the new query to the opcode tree.

11. The filter engine as recited in claim 10, the opcode merger further configured to traverse the opcode tree to determine if an opcode node corresponding to the new query already exists in the opcode tree and add new opcode nodes that correspond to query opcode blocks that are not already included in the opcode tree.

12. The filter engine as recited in claim 10, wherein opcode nodes corresponding to opcode blocks included in a common prefix are represented as a shared segment in the opcode tree.

13. The filter engine as recited in claim 10, wherein queries are merged into the opcode tree dynamically at runtime.

14. The filter engine as recited in claim 10, further comprising XPath queries in the plurality of queries.

15. The filter engine as recited in claim 10, the compiler being further configured to create opcode objects that are configured to merge themselves into an appropriate location in the opcode tree.

16. The filter engine as recited in claim 10, the opcode merger being further configured to perform the following steps:

when an opcode node will depend from a branch node when added to the opcode tree, identifying one or more child opcode nodes that depend from the branch opcode; and

implementing an optimized branch node that includes an optimized indexed lookup procedure if such implementation would increase branch processing efficiency and referencing the opcode node from the optimized branch node.

17. The filter engine as recited in claim 16, wherein the optimized indexed lookup procedure further comprises a procedure selected from the following list: a hash function; a tries function; an interval tree function.

18. The filter engine as recited in claim 16, wherein the opcode merger is further configured to restore an optimized branch node to a generic branch node when the optimized branch node is no longer more efficient that the generic branch node.

19. A compiler stored on one or more computer-readable media containing computer-executable instructions for performing the following steps:

receiving a query to be added to an opcode tree that represents a plurality of queries, at least two of which include similar prefixes; and

compiling a query to produce one or more opcode objects that are each configured to merge into the opcode tree as an opcode node by determining an appropriate location in the tree to merge, and merging into the tree in accordance with a node context of the appropriate location.

20. The compiler as recited in claim 19, further comprising producing opcode objects that are further configured to merge into the opcode tree only if an identical opcode object corresponding to a similar query prefix is not already included in the opcode tree

21. The compiler as recited in claim 19, wherein a query further comprises an XPath query.

22. The compiler as recited in claim 19, the opcode object being further configured to perform the following steps:

determining a function that the opcode object performs;

determining if a branch node that will reference the opcode node corresponding to the opcode object also references other opcode nodes that perform a similar function; and

implementing an optimized branching function in the branch node if the branch node can be optimized to more efficiently process the opcode nodes that it references.

23. The compiler as recited in claim 22, wherein an optimized branching function further comprises a function selected from the following list of functions: a hash function, an interval tree function; a function utilizing tries.

24. The compiler as recited in claim 22, wherein the branch node is configured to recognize a context where the optimized branching function is no longer efficient and to resort to its previous function if such a context develops.

25. The compiler as recited in claim 19, wherein:

the compiler is configured to receive the query and generate the opcode at runtime; and

the opcode node is configured to merge itself into the opcode tree at runtime.

26. An opcode object stored on one or more computer-readable media including computer-executable instructions that, when executed on a computer, perform the following steps:

determining an appropriate location to merge itself as a new opcode node in an opcode tree when a query from which the opcode object is derived is added to a filter table represented by the opcode tree including opcode nodes that, when executed, evaluate the queries;

evaluating a node context of the location to which the new opcode node will be added; and

merging itself into the opcode tree by adding and/or modifying references from an opcode node or a branch node to the new opcode node.

27. The opcode block as recited in claim 26, further configured to perform the recited steps dynamically at runtime.

28. The opcode block as recited in claim 26, further configured to perform the recited steps within a .NET environment.

29. The opcode block as recited in claim 26, wherein evaluating a node context further comprises:

identifying a generic branch opcode from which the new node will depend;

identifying one or more other nodes that depend from the generic branch opcode that include a similar expression as the new node; and if a sufficient number of the one or more other nodes exists, modifying the generic branch opcode to an optimized branch opcode that is optimized to more efficiently process the similar expressions.

30. The opcode block as recited in claim 26, wherein evaluating a node context further comprises:

identifying an optimized branch opcode from which the new node will depend;

identifying one or more other nodes that depend from the optimized branch opcode that include a similar expression as the new node; and

if minimum threshold number of the one or more other nodes is not met, modifying the optimized branch opcode to a generic branch opcode that can process the number of one or more other nodes more efficiently than the optimized branch opcode can.

31. A method for removing a first query from an opcode tree, comprising:

identifying an opcode tree that includes opcode nodes representing multiple queries such that when the opcode tree is executed, each of the multiple queries is evaluated;

identifying one or more opcode nodes that correspond to the first query; and

removing any opcode node that does not correspond to a second query.

32. The method as recited in claim 31, further comprising the step of modifying a branch node that references an opcode node that is removed from the opcode tree.

33. The method as recited in claim 32, wherein the modifying further comprises removing the branch node if the branch node references only one other opcode node other than the opcode node to be removed.

34. The method as recited in claim 32, wherein the modifying further comprises removing an optimized lookup function from the branch node if removing the branch node renders the lookup function less efficient that a direct comparison function.

35. The method as recited in claim 32, wherein the modifying further comprises implementing an optimized processing function in the branch node if the removal of the branch node creates a context in which the optimized processing function would increase efficiency of the branch node processing.

36. The method as recited in claim 35, wherein the optimized processing function further comprises one of the following functions: a hash function; an interval tree function; a function using tries.

37. One or more computer-readable media containing computer-executable instructions that, when executed on a computer, perform the following steps:

identifying an opcode block that corresponds to a query to be added to an opcode tree that represents multiple queries with a plurality of opcode nodes;

identifying an appropriate location in an opcode tree to situate new opcode nodes that correspond to a sequence of opcode objects in the opcode block, the opcode tree including at least one shared opcode node that corresponds to at least two of the multiple queries;

evaluating a location context; and

modifying an opcode node or a branch node to incorporate a new opcode node.

38. The one or more computer-readable media as recited in claim 37, the evaluation step further comprising: evaluating a plurality of dependent opcode nodes that depend from a branch node from which the new opcode node will depend; and

the modifying step further comprising modifying the branch node to include an indexed lookup function if the dependent opcode nodes perform a similar function and processing the dependent opcode with the indexed lookup function increases the efficiency thereof.

39. The one or more computer-readable media as recited in claim 38, the indexed lookup function further comprising a hash function, a tries function or an interval tree function.

40. The one or more computer-readable media as recited in claim 37, wherein the queries are XPath queries.

41. The one or more computer-readable media as recited in claim 37, wherein the steps are performed by an inverse query engine.

42. The one or more computer-readable media as recited in claim 37, wherein the identifying step, the evaluating step and the modifying step are performed by the new opcode node.

43. The one or more computer-readable media as recited in claim 37, wherein the steps are performed in a Common Language Runtime (CLR) environment.